universität wien

Bachelorarbeit

Titel der Bachelorarbeit

„Darstellung von Kundenverhalten mittels einer Android App und Daten aus Perceptrons“

Verfasser

Nicolas Mattersdorfer 11807110

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2024

Studienkennzahl lt. Studienblatt:	A 033 526
Fachrichtung:	Wirtschaftsinformatik
Betreuerin / Betreuer:	Dipl.-Ing. Marian Lux

Eidesstattliche Erklärung

Ich Nicolas Mattersdorfer Student an der Universität Wien (Matrikelnummer: 11807110) erkläre eidesstattlich, dass ich die Arbeit selbständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und alle aus ungedruckten Quellen, gedruckter Literatur oder aus dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte gemäß den Richtlinien wissenschaftlicher Arbeiten zitiert, durch Fußnoten gekennzeichnet bzw. mit genauer Quellenangabe kenntlich gemacht habe. Diese schriftliche Arbeit wurde noch an keiner anderen Stelle vorgelegt.

13.02.2024 Nicolas Mattersdorfer

Datum Unterschrift Studierender

Abstract

Diese Bachelorarbeit wurde an der Universität Wien im Bereich Wirtschaftsinformatik geschrieben und beschäftigt sich mit der Visualisierung von Perceptron-Daten in einem Android Projekt. Für das Projekt wurde ein trainiertes Model verwendet und aus den Ergebnissen und erhaltenen Daten wurde eine Android-Applikation entwickelt. Diese Daten wurden aufbereitet und dem User in verschiedenen Kategorien und Übersichten zur Verfügung gestellt. Dadurch soll eine bessere Lesbarkeit für den User generiert werden. Zusätzlich zu dem praktischen Projekt soll die Bachelorarbeit den theoretischen Hintergrund zu Process Mining und Perceptrons erklären.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

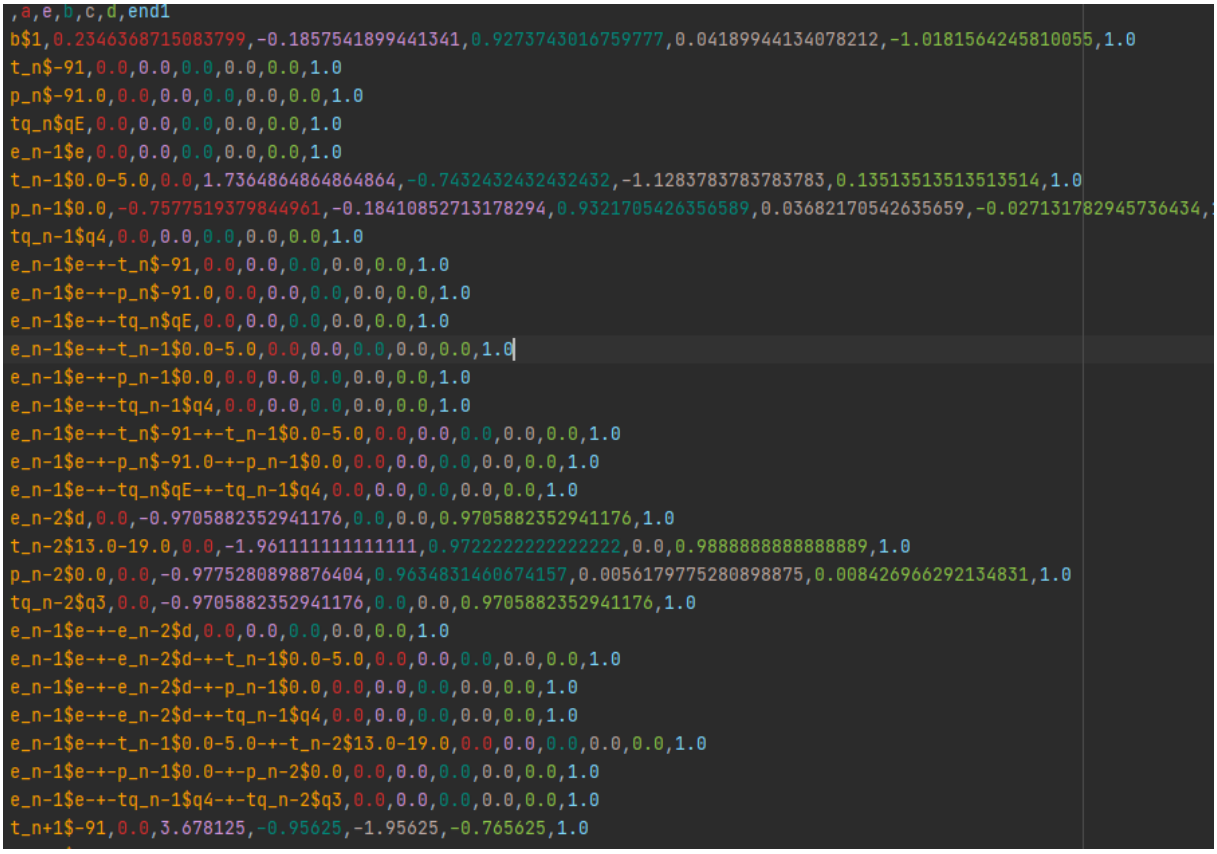
In der modernen Zeit wird die Optimierung von Daten und Kundenverhalten immer wichtiger, um Produkte und Dienstleistungen gewinnbringend zu vermarkten. Viele dieser neuen Applikationen oder Produkte werden so konzipiert, dass sie über eine lange Laufzeit optimiert werden. Durch diese Optimierung ist es möglich, den Informationsgehalt von Daten zu erhöhen. Das Ziel dieser Bachelorarbeit ist es, sich genau mit diesen theoretischen und praktischen Hintergründen zu beschäftigten und die Theorie in der Praxis anzuwenden.

Im Rahmen dieser Bachelorarbeit wird zuerst ein kleiner theoretischer und praktischer Hintergrund gegeben und anschließend die Struktur und Umsetzung der Theorie in einer Android-Applikation wiedergegeben. Für den Theorie-Aspekt wird auf Process Mining in Kombination mit Perceptrons eingegangen und ein grober Überblick gegeben. Diese gewonnenen Informationen und Theorie-Aspekte helfen dabei, die Struktur und den Aufbau der Android Applikation zu bilden und anschließend die notwendigen Daten zu verwenden und anzureichern. Somit beschäftigt sich der praktische Teil mit der Umsetzung der Theorie-Aspekte auf echte Datensätze.

1.1. Ausgangslage

Es gibt schon eine Applikation für IOS, die einige der angestrebten Funktionen beinhaltet, aber nicht alle, die implementiert werden sollen. Die alte Applikation sollte vom Konzept angepasst und in Android aufgebaut werden. Hierzu ist eine Menge an Daten und Strukturen notwendig. Die Grunddaten wurden am Start der Bachelorarbeit zur Verfügung gestellt und mussten verwendet werden. Diese Daten bilden ein fertig trainiertes Perceptron und die Werte und Kategorien sind die Grunddaten für die Struktur der Android Applikation. Diese Daten wurden in einem CSV-File bereitgestellt und sollten auch so verwendet werden, um in Zukunft neue Daten durch das Programm zu jagen.

Als Beispiel für diese Strukturen und das File dient dieser CSV-Ausschnitt.



```
,a,e,b,c,d,end1
b$1,0.2346368715083799,-0.1857541899441341,0.9273743016759777,0.04189944134078212,-1.0181564245810055,1.0
t_n$-91,0.0,0.0,0.0,0.0,0.0,1.0
p_n$-91.0,0.0,0.0,0.0,0.0,0.0,1.0
tq_n$qE,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e,0.0,0.0,0.0,0.0,0.0,1.0
t_n-1$0.0-5.0,0.0,1.7364864864864864,-0.7432432432432432,-1.1283783783783783,0.13513513513513514,1.0
p_n-1$0.0,-0.7577519379844961,-0.18410852713178294,0.9321705426356589,0.03682170542635659,-0.027131782945736434,
tq_n-1$q4,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-t_n$-91,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-p_n$-91.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-tq_n$qE,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-t_n-1$0.0-5.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-p_n-1$0.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-tq_n-1$q4,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-t_n$-91-+-t_n-1$0.0-5.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-p_n$-91.0-+-p_n-1$0.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-tq_n$qE-+-tq_n-1$q4,0.0,0.0,0.0,0.0,0.0,1.0
e_n-2$d,0.0,-0.9705882352941176,0.0,0.0,0.9705882352941176,1.0
t_n-2$13.0-19.0,0.0,-1.961111111111111,0.9722222222222222,0.0,0.9888888888888889,1.0
p_n-2$0.0,0.0,-0.9775280898876404,0.9634831460674157,0.0056179775280898875,0.008426966292134831,1.0
tq_n-2$q3,0.0,-0.9705882352941176,0.0,0.0,0.9705882352941176,1.0
e_n-1$e-+-e_n-2$d,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-e_n-2$d-+-t_n-1$0.0-5.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-e_n-2$d-+-p_n-1$0.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-e_n-2$d-+-tq_n-1$q4,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-t_n-1$0.0-5.0-+-t_n-2$13.0-19.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-p_n-1$0.0-+-p_n-2$0.0,0.0,0.0,0.0,0.0,0.0,1.0
e_n-1$e-+-tq_n-1$q4-+-tq_n-2$q3,0.0,0.0,0.0,0.0,0.0,1.0
t_n+1$-91,0.0,3.678125,-0.95625,-1.95625,-0.765625,1.0
```

Abbildung 1: CSV-Grunddaten

Hier ist ersichtlich, dass die Datei in folgender Reihenfolge aufgebaut ist. Die erste Reihe bildet die Knotenpunkte am Perceptron ab. Diese haben dann in den folgenden Reihen immer ein „Event“ plus einen dazugehörigen Wert. Hierzu kann man sagen, dass die Events in 4 erkennbare Identifier unterteilbar sind, welche auch noch unterteilt sind.

Diese Identifier sind:
T_N$ - Temperaturwerte, die aufgetreten sind.
P_N$ - Wetterbedingungen, ob es geregnet hat oder nicht.
TQ_N$ - Tageszeit wo das Event stattgefunden hat.
E_N$ - Aktivität die vorrangegangen oder nachfolgend ist.

Diese Werte können auch variieren, wie T_N-1$ wäre die Temperatur des vorherigen Events und T_N+1$ wäre die Temperatur des nachfolgenden Events. Diese können bis +2 und -2 gehen und bilden somit die Grundstruktur, die für den Aufbau der Daten in der Applikation essenziell sind.

Die verschiedenen Identifier können auch kombiniert werden. In diesem Fall werden die Identifier mit der Zeichenkette -+- verbunden. Dies ist auch in der Grafik ersichtlich und bildet eine sehr wesentliche Funktion. Durch diese Verbindung wird es möglich, komplexere Events abzubilden und daraus logische Schlüsse zu ziehen. Ein simples Beispiel wäre hier eine Kombination der Tageszeit mit den Wetterbedingungen. Die Verbindung P_N$ mit TQ_N$ bildet somit die Kombination aus der Tageszeit 12:00-18:00 und der Wetterbedingungen Niederschlag. Wenn diese Kombination eintritt, kann es wahrscheinlicher sein das Menschen eher online shoppen als bei gutem Wetter um diese Uhrzeit. Durch diese Kombinationen ist es somit möglich bessere Schlüsse aus dem Verhalten der Menschen zu ziehen.

2. Theorie zu Perceptron und Process Mining

Doch warum sind diese Werte, die im letzten Kapitel genannt wurden, überhaupt relevant und wie sind sie durch das Perceptron entstanden? Was ist ein Perceptron und was ist Process Mining? Diese Fragen werden in diesem Kapitel behandelt und genauer auf die Geschichte der Entstehung und Verwendung eingegangen.

2.1. Perceptron

Perceptrons sind Teil des großen Gebietes des Machine Learning. Dieses Kapitel soll zuerst die generelle Funktionsweise eines Menschlichen Neurons erklären, da dies der wesentliche Bestandteil eines Perceptrons ist. Im Anschluss wird die Entstehung und Funktionsweise eines Perceptrons dargestellt und erklärt.

2.1.1. Neuronen

Der Grundbestandteil jedes Perceptrons sind biologische Neuronen. Diese Neuronen wurden in den 50iger Jahren von Frank Rosenblatt zu künstlichen Neuronen weiterentwickelt und angepasst, um die biologischen Neuronen im Gehirn eines Menschen widerzuspiegeln. (vgl. Rosenblatt, 1958)

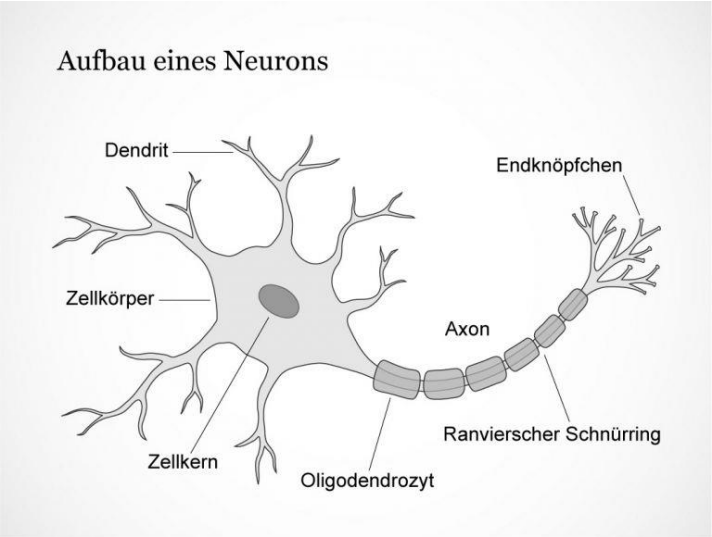


Abbildung 2: Menschliches Neuron

Die Grundfunktionsweise ist nicht sonderlich kompliziert und beruht auf elektrischen Signalen. Der Zellkörper nimmt elektrische Signale über die außenstehenden Dendriten auf und trifft anhand der Verarbeitung im Zellkörper Entscheidungen, welche über das Axon weitergegeben werden. Diese Struktur inspirierte Frank Rosenblatt aus diesem menschlichen Neuron ein künstliches zu entwickeln.

2.1.2. Entstehung und Funktionsweise von einem Perceptron

Ein Perceptron kann eine Kombination von einem oder mehreren künstlichen Neuronen sein. (vgl. Rosenblatt, 1958)

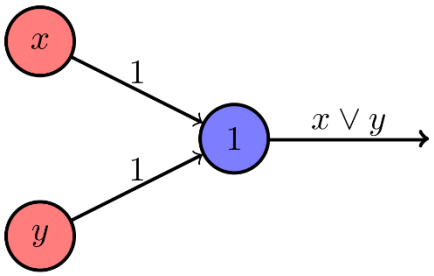


Abbildung 3: Logisches Oder Perceptron

Diese Grafik zeigt beispielsweise ein logisches Oder in einem Perceptron. Es gibt zwei Input Neuronen die einen Wert an das Output Neuron weitergeben und eine Gewichtungsfunktion, die den Wert vergleicht und einen Output ausgibt. Das Output Neuron schaltet in diesem Fall nur auf eins, wenn mindestens einer der beiden Werte der Inputs auf eins steht. Dieses Perceptron wird auch Single Layer Perceptron genannt da das Perceptron nur aus einem Layer besteht.

Single Layer Perceptrons waren anfänglich für viele Problemstellungen ausreichend. Die verschiedenen Gewichtungsfunktionen konnten nur linear trennbare Probleme lösen und auswerten. Hier kam man aber relativ schnell zu der Erkenntnis, dass dies nicht ausreichend ist, da auch nicht linear trennbare Problemstellungen auftraten. Das klassische Beispiel für nicht linear trennbare Probleme ist das XOR-Problem. (vgl. Sabry, 2023)

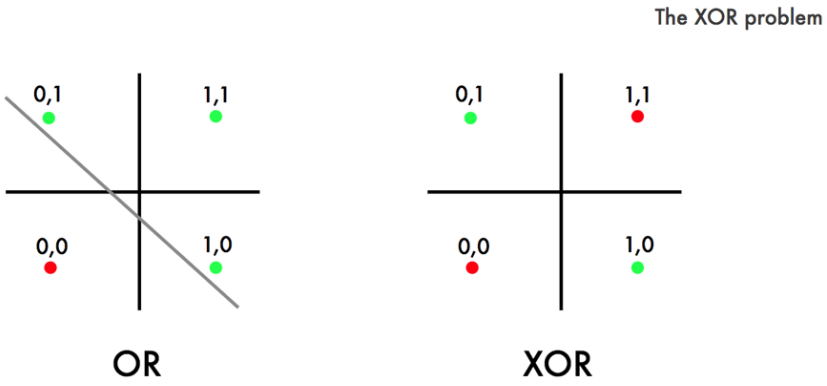


Abbildung 4: XOR-Problem

Aus der Grafik ist leicht ersichtlich, warum das XOR-Problem so gravierend ist. Bei einem logischen Oder ist es leicht die Werte auf einer linearen Basis zu trennen und so die Werte richtig zu klassifizieren. Bei einem XOR ist dies nicht möglich da die Werte nicht linear mit einer Linie trennbar sind. (vgl. Nitta, 2003)

Durch die intensive Forschung in diese Richtung entstanden neuronale Netzwerke mit mehreren Layer. Ein Layer ist eine Ebene, auf der sich das Perceptron befindet und Daten an die nächste Ebene übergibt, wo sie wieder gebündelt werden und neuen Kategorien zugeteilt werden. Durch den zusätzlichen Layer war es möglich, komplexere Aufgaben und Klassifizierungen durchzuführen. Durch diese Änderungen ist es in der Regel möglich, lineare und nicht-lineare Probleme zu identifizieren und zu lösen. (vgl. Sabry, 2023)

Das XOR-Problem lässt sich also mit einem mehrschichtigen Perceptron beheben. Die folgende Grafik zeigt, wie das Problem gelöst wird.

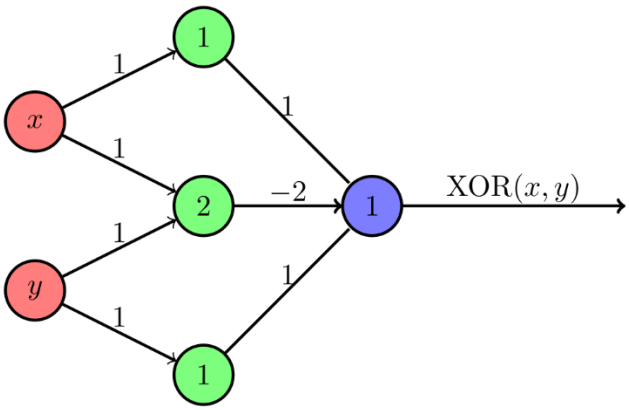


Abbildung 5: XOR-Problem gelöst

Diese Grafik ist eine simple Darstellung eines mehrschichtigen Perceptron. Es ist leicht ersichtlich, dass die Werte am Anfang in den Input Layer gehen und durch die Gewichtung in den Hidden Layer eintreten. Hier werden die Werte immer weiter verändert und angepasst und dann an den Output Layer übergeben. Die Werte werden in der Regel gewichtet. Das heißt, dass die Werte mit einer bestimmten Zahl verändert werden, um am Ende in der Zielfunktion eine größere oder niedrigere Wichtigkeit zu erhalten. Durch diese Gewichtungen und den Hidden Layer war es möglich das XOR-Problem zu lösen.

Diese Modelle und Gewichtungen sind in der Regel bei neuen Problemen aber nicht immer optimal. Dazu gibt es verschiedene Methoden, wie man diese ändern und verbessern kann. Zwei dieser Methoden sind Forward Propagation und Backward Propagation beziehungsweise Vorwärtsausbreitung und Rückwärtsausbreitung. Bei dieser Methode werden die Werte zuerst durch das System gewichtet und zum Output Layer transportiert. Am Ende wird überprüft, ob das gewünschte Ergebnis erreicht wurde und sollte dies nicht der Fall sein, wird pro Layer die Gewichtung angepasst und dann erneut versucht. Dies wird wiederholt, bis das neuronale Netzwerk die gewünschten Werte ausgibt und dadurch das Ergebnis nicht verfälscht ist. (vgl. Trask, 2019)

In der heutigen Zeit gib es eine Vielzahl an verschiedenen Möglichkeiten wie man Perceptrons verwenden kann. Diese haben immer unterschiedliche Anwendungsmöglichkeiten, bei welchen sie besser oder schlechter funktionieren und werden dementsprechend in verschiedenen Feldern des Machine Learning

angewendet. Für die Arbeit im speziellen wurde ein Linear Averaged Multiclass Perceptron verwendet.

2.2. Process Mining

Process Mining war ein Begriff, der vor gut 20 Jahren entstand, also rund um die Jahrtausendwende und seitdem eine sehr große Relevanz aufgebaut hat. Einer der Forscher, welcher diese Technologie mitgeprägt hat, war Professor Will van der Aalst, der einen großen Beitrag daran hatte, dass diese Konzepte und Arbeitsweisen sich in der modernen Technik etabliert haben. In der modernen Zeit wird Process Mining verwendet, um aus verschiedensten Rohdaten neue und aufbereitete Informationen zu gewinnen, um sie anschließend zu analysieren und für den Menschen visuell darzustellen. Diese Methoden werden sehr oft für Geschäftsprozesse verwendet. Durch die gewonnenen neuen Informationen können die Geschäftsprozesse visuell dargestellt werden und sind somit leichter zu identifizieren. Sollte es zu Abweichungen vom gewünschten Ergebnis kommen, kann man als Unternehmen eingreifen und somit das Geschäft optimieren. (vgl. van der Aalst, 2016)

Wie funktioniert nun diese Optimierung konkret? Grundlegend ist zu Process Mining zu sagen, dass in der Regel immer verschiedene Eventpunkte beziehungsweise Ereignisdaten mit unterschiedlichen Methoden überprüft, verändert oder angepasst werden. Durch diese Veränderung der Daten werden sie aufbereitet und ergeben ein Gesamtbild, aus dem später Schlüsse gezogen werden können. Diese Eventdaten folgen in der Regel einer zeitlichen Abfolge und können so identifiziert werden und einer Zeitlinie zugeordnet werden. Aus dieser Linie kann dann eine Visualisierung erfolgen. (vgl. van der Aalst, 2016)

2.2.1. Hauptaufgaben von Process Mining

Process Mining beschäftigt sich hauptsächlich mit drei verschiedenen Aufgaben: Prozessentdeckung, Konformitätsprüfung und Prozessoptimierung. Dieses Kapitel soll den Kerngedanken dieser drei Techniken wiedergeben und erklären. (vgl. van der Aalst, 2014)

Prozessentdeckung

Die erste Phase des Process Mining beschäftigt sich mit der konkreten Prozessentdeckung. In dieser Phase sollen die Prozessmodelle aus den vorhandenen Daten oder Strukturen extrahiert und differenziert werden. Diese extrahierten Modelle bilden also die konkret gesammelten Daten des Unternehmens oder des Prozesses. Um diese Prozesse zu entdecken können verschiedenste Algorithmen verwendet werden. Einige Beispiele wären der Alpha-Algorithmus, Heuristisches Mining oder Genetisches Prozess Mining. (vgl. van der Aalst, 2014)

Konformitätsprüfung

Die Konformitätsprüfung beschäftigt sich mit der Validierung der erstellten Modelle und den realen Daten. Dies ist notwendig, um eine Vollständigkeit und Richtigkeit der Modelle sicherzustellen und somit eine gute Ausgangsbasis für Analysen und Verbesserungen zu bilden. Dieser Arbeitsschritt ist aber nicht immer einfach, da sich oft aus den vorhandenen Daten keine perfekt nachgebauten Modelle bilden lassen und somit die Überprüfung der Werte und Modelle nicht immer zu einem positiven Ergebnis führt. Dadurch muss das Modell verändert oder neu generiert werden. (vgl. van der Aalst, 2014)

Prozessoptimierung

Die Prozessoptimierung bringt den eigentlichen Mehrwert für das Prozess Mining, da hier, durch die Änderung an bestehenden Prozessen ein Plus für Unternehmen oder Organisationen erwirtschaftet werden kann. Hierfür werden verschiedenste Verbesserungsmöglichkeiten analysiert und durch eine Neugestaltung oder Umgestaltung verbessert. (vgl. van der Aalst, 2014)

Diese drei Aufgaben sind nicht immer komplett in einem Durchlauf abschließbar. Da zum Beispiel bei gefundenen Fehlern in der Konformitätsprüfung wieder ein neues Modell in der Process Discovery erstellt werden muss und der Prozess wieder von vorne beginnen kann.

2.2.2. Use Cases

Bei Process Mining gibt es verschiedene Miner welche verwendet werden. Auf drei konkrete Anwendungsfälle wird nun kurz genauer eingegangen und erklärt, wie sie

funktionieren. Die Beispiele aus diesem Kapitel sind aus dem Buch Process Mining Data Science in Action. (vgl. van der Aalst, 2016)

Alpha Miner

Beim Alpha Miner werden die Eventlogs in eine zeitliche Sequenz geschalten, um ein Netz zu bilden. Diese Netze, auch Petri Netze genannt, zeigen die Daten und Entscheidungen in einer zeitlichen Abfolge und können durchlaufen werden. Durch diese Visualisierung ist es möglich, verschiedene Szenarien zu sehen und Entscheidungen durchzuspielen.

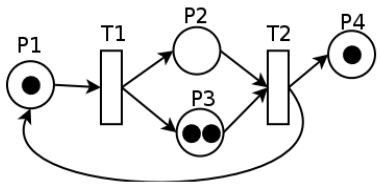


Abbildung 6: Alpha Miner

Die Idee ist, die Beziehungen von Events zu Ordnen. Es gibt vier verschiedene Beziehungen, die geordnet werden können:

Direct Succession bedeutet das in einem Fall X > Y das X der Vorgänger von Y ist also das Y auf X folgt.

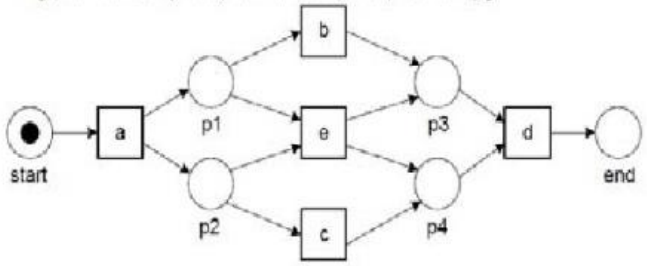
Causality stellt den Fall dar, dass X > Y aber nicht den Fall Y > X. Daraus schließt man X → Y.

Parallel trifft zu, wenn X > Y und X > Z, dann wäre Y || Z parallel.

Unrelated ist eine Beziehung, wenn es keine Fälle von X > Y oder Y < X gibt. Dann wäre X # Y keine Verbindung.

Diese einzelnen Schritte führt man für alle Beziehungen durch und fügt sie einer Liste hinzu. Durch diese Liste kann man eine Tabelle erstellen, in welcher alle Fälle eingetragen sind und die Häufigkeit ersichtlich ist.

$$L_1 = [\langle a,b,c,d\rangle^3, \langle a,c,b,d\rangle^2, \langle a,e,d\rangle]$$



	a	b	c	d	e
a	$\#_{L_1}$	$\rightarrow_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$
b	$\leftarrow_{L_1}$	$\#_{L_1}$	$\|\|_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
c	$\leftarrow_{L_1}$	$\|\|_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
d	$\#_{L_1}$	$\leftarrow_{L_1}$	$\leftarrow_{L_1}$	$\#_{L_1}$	$\leftarrow_{L_1}$
e	$\leftarrow_{L_1}$	$\#_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$

Abbildung 7: Alpha Miner Liste und Net

Aus dieser Liste kann man nun das Petri Netz extrahieren und die einzelnen Punkte darstellen. Wenn man sich den Punkt B ansieht, ist ersichtlich das nur eine Verbindung zum Punkt B hinführt, Punkt A. Dies führt man für jeden Punkt durch und erstellt anschließend das fertige Petri Netz.

Um zu veranschaulichen, wie aus einer Beziehung ein Petri Netz entsteht, werden hier einige Beispiele beschrieben:

X→Y

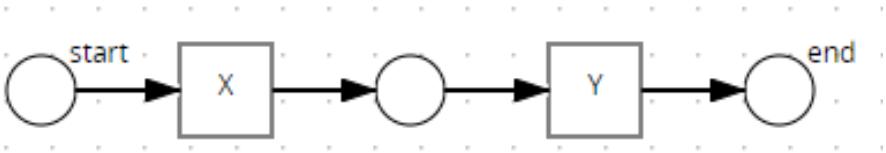


Abbildung 8: Nach X folgt Y

X→Y, X→Z, Y||Z

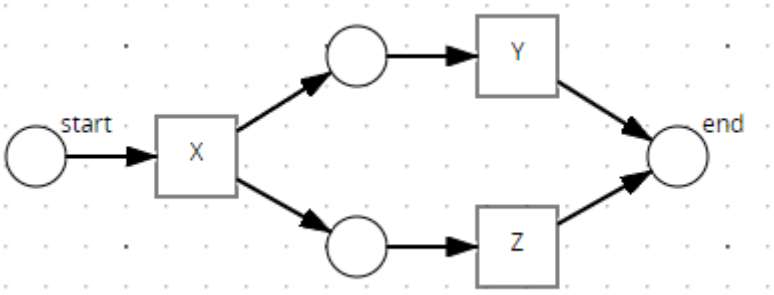


Abbildung 9: Nach X folgt Y und Z

X→Z, Y→Z , X||Y

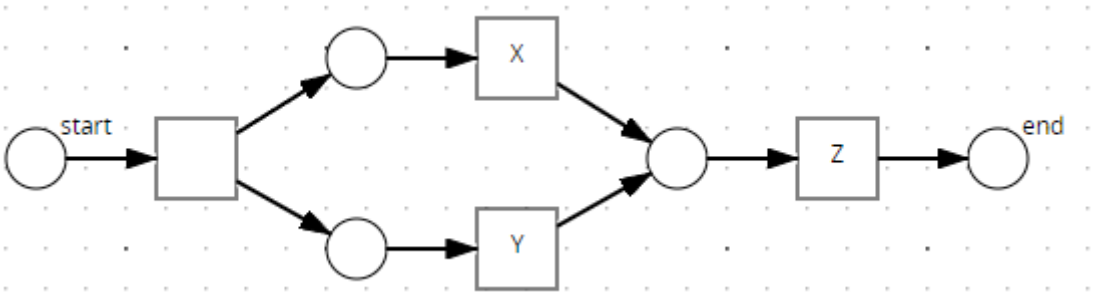


Abbildung 10: Nach X und Y folgt Z

Aus diesen einzelnen Bausteinen wird aus der Tabelle ein fertiges Petri Netz generiert. Dieses Petri Netz spiegelt die einzelnen Prozessschritte wider und kann somit getestet und optimiert werden.

Heuristic Miner

Der Heuristic Miner fügt zu jedem Pfad in einem Graphen einen bestimmten Wert hinzu. Diese Werte repräsentieren die Häufigkeit der Aufrufe der Pfade. Durch diese Häufigkeit ist es leichter, falsche beziehungsweise schlechte Pfade zu ermitteln und diese zu eliminieren.

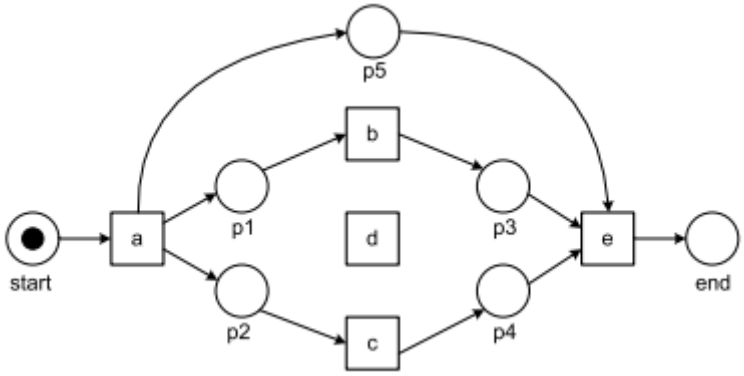


Abbildung 11: Heuristic Miner unbefüllt

Die Abbildung zeigt einen Heuristic Miner ohne Gewichtungen. Um die Gewichtungen zu kalkulieren, muss man jeden einzelnen Pfad zuerst Evaluieren und die Anzahl der eingetretenen Ereignisse zusammenrechnen. Dies bildet den Direct Successor Count, welcher benötigt wird, um die Dependency Measure zu berechnen. Die Dependency Measure ist notwendig, um die Pfade zu gewichten. Für diese Dependency Measure gibt es zwei Formeln. In den folgenden Formeln wird der Direct Successor Count als DCA bezeichnet und Dependency Measure als DM, um die Lesbarkeit zu verbessern.

Wenn es nur in eine Richtung geht, also wie oben in dem Beispiel von A nach B dann gilt, folgende Formel für A => B:

$$DM = DCA\ |A > B| / (DCA\ |A > B| + 1)$$

Wenn es aber konkurrierende Pfade gibt, beispielsweise wenn der Weg von B nach A auch möglich ist, gilt folgende Formel für A => B:

$$DM = DCA\ |A > B| – DCA\ |B > A| / (DCA\ |A > B| + DCA\ |B > A| + 1)$$

Die Dependency Measure kann zwischen -1 und 1 sein. Wenn 0 vorhanden ist gibt es keine Dependency. Das Spektrum Richtung -1 gibt an das eher in die andere Richtung eine Abhängigkeit besteht. Also wenn A > B den Wert -1 hätte dann würde das bedeuten das öfter B > A besteht.
Wenn jeder dieser DCAs berechnet wurde, kann man daraus den gewichteten Graphen erstellen.

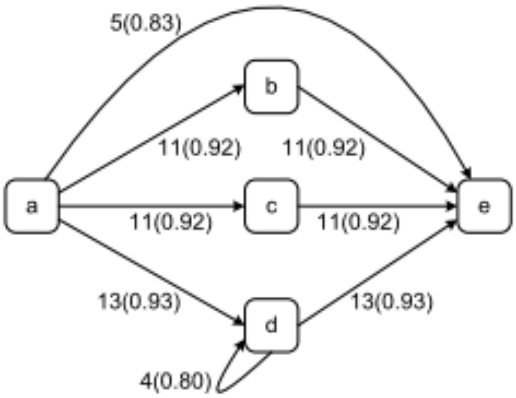


Abbildung 12: Heuristic Miner befüllt

Um dies kurz zu veranschaulichen nun ein kurzes Beispiel:

DCA |A > B| hätte einen Count von 12
DCA |B > A| hätte einen Count von 10

DM für A > B wäre hier: 12 – 10 / 12+10+1 = 0.6666
DM für A < B wäre hier: 10 – 12 / 10+12+1 = -0.6666

Aus diesem kurzen Beispiel ist ersichtlich das die Verbindung A > B einen positiven Wert hat und die Verbindung A < B einen negativen Wert. Daraus kann man schließen, dass die Verbindung A > B öfter vorkommt. Im nächsten Schritt kann man den dazugehörigen Graphen erstellen und die Gewichtung einfügen.

Inductive Miner

Der Inductive Miner ist eine spezielle Form, da hier auch neue und mögliche Pfade erkannt und getestet werden. Es werden durch einen Algorithmus neue Knotenpunkte und Möglichkeiten iterativ entdeckt und getestet. Er bietet eine hohe Flexibilität und kann ein breites Spektrum abdecken.

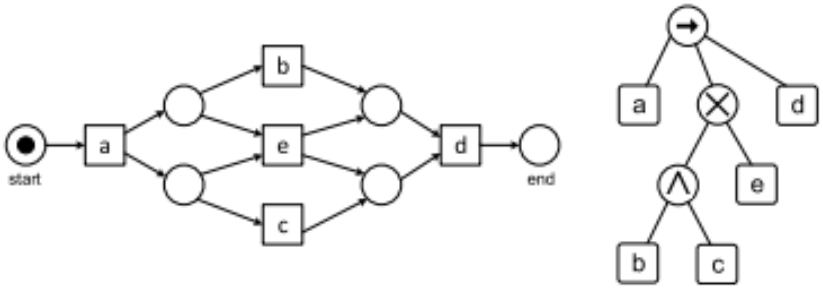


Abbildung 13: Inductive Miner

Die Grafik zeigt den Process Tree eines Inductive Miner. In dieser Grafik sieht man verschiedene Aktivitäten. In diesem Fall wäre a eine normale Aktivität, die auf dem sequenziellen Ablauf ist. Auf der rechten Seite sieht man die logische Struktur dahinter. Der Pfeil beschreibt die Abfolge von Aktivitäten. Das X beschreibt ein XOR welches durchgeführt wird und das ∧ Symbol beschreibt ein Parallel.
Der Inductive Miner startet generell mit einem Datenset aus Event Logs. Diese Event Logs zeigen welche Abhängigkeiten zwischen den einzelnen Events bestehen und wie die Reihenfolge der einzelnen Elemente ist. Diese Elemente werden dann in Reihenfolge in das Prozessmodell integriert und hinzugefügt.

Beispiel zur Erstellung von einem Process Modell mit dem Inductive Miner:
Der Event Log sieht wie folgt aus: [[A,B,C],[A,B,D],[C,E],[B,D],[A,B]]
Aus diesem Event log wird ein Petri Netz erstellt.

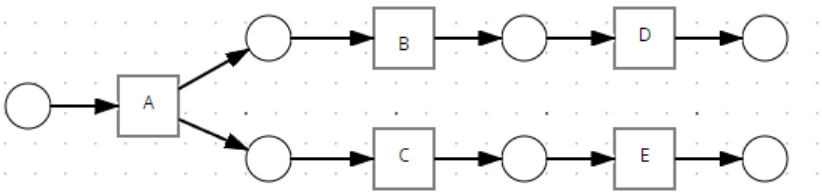


Abbildung 14: Petri Netz Inductive Miner

Aus diesem Petri Netz wird ein Workflow Net erstellt und es können verschiedene Cuts beziehungsweise Schnitte durchgeführt werden. Diese Schnitte sind dafür da, einzelne Teilaspekte des Modells zu analysieren und diese zu vergleichen, ohne das gesamte System zu analysieren.

2.3. Zielsetzung

Das Ziel dieser Bachelorarbeit ist es eine Android-Applikation zu entwickeln, welche es ermöglicht gegebene CSV-Dateien einzulesen und anschließend den Inhalt dieser Dateien grafisch und in Form von Listen für den User ersichtlich zu machen. Der Inhalt der CSV-Datei ist ein trainiertes Perceptron welches dargestellt und analysiert werden soll.

3. Projektdokumentation

Dieses Kapitel soll die Ergebnisse der Implementierung wiedergeben und eine Dokumentation der einzelnen Klassen und wichtigen Funktionen darstellen.

3.1. Verwendete Technologien

Bei der Wahl des Betriebssystems gab es freie Auswahl. Zur Auswahl standen IOS oder Android. Die bestehende Applikation läuft auf IOS und Apple Geräten. Um eine neue Software zu entwerfen und nicht an der alten Version festzuhalten, wurde Android gewählt. Hier gibt es sehr viele Versionen und es wurde sich für Android 8.1 Oreo entschieden, da diese eine Abdeckung von 90.2 % der am Markt vorhandenen Geräten bereitstellt.

Das komplette Projekt wurde in Android Studio entwickelt und getestet. Für das Testen wurde ein virtuelles Gerät verwendet, da kein Android-Gerät in realer Form zur Verfügung stand. Es wurde auf zwei verschiedenen Devices, namentlich Pixel 2 XL API 30 R und Pixel 2 API 27 Oreo getestet und programmiert.

3.2. Implementierung und Ergebnisse

Die Voraussetzungen und Ziele der Bachelorarbeit konnten bis auf die visuelle Darstellung des Perceptron komplett erfüllt werden. Auf dieses Problem wird später bei den Kapitel Problemen genauer eingegangen. Für den Rest der Applikation ist es

möglich, eine CSV-Datei einzulesen und die Daten dann dynamisch in verschiedenen Aktivitäten einzusehen. Diese Aktivitäten werden als Node Punkte bezeichnet. Diese Nodes repräsentieren einen Schritt in dem Perceptron. Das folgende Kapitel soll die einzelnen Seiten und Funktionen darstellen und beschreiben.

3.3. Struktur des Projektes

Dieses Kapitel soll auf die Struktur der einzelnen Klassen und Files eingehen und das Projekt dokumentieren. Das Projekt hat zwei verschiedene Säulen. Die erste Säule ist der Java Code mit den einzelnen Aktivitäten und der Programmlogik. Die zweite Säule sind die einzelnen Grafik-Ressourcen für die Visualisierung und die Applikationskomponenten. Die folgende Grafik zeigt die gesamte Projektstruktur der relevanten Klassen und Files.

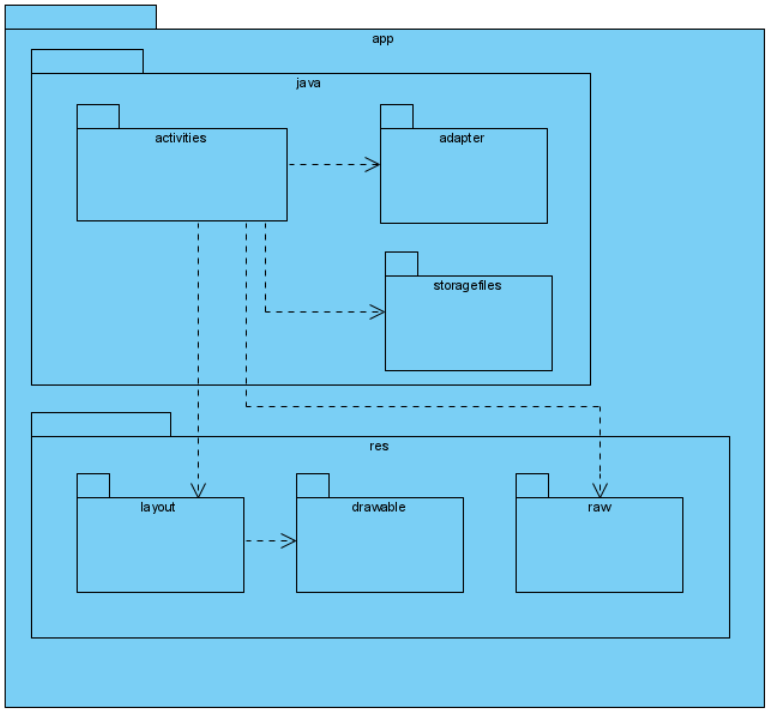


Abbildung 15: Projektstruktur

Der Hauptkern des Projektes besteht aus den activities Paket der Android Applikation. Hier werden die einzelnen Daten verarbeitet, geladen und dargestellt. Um die Daten persistent für alle Activities zu speichern, gibt es ein eigenes Paket storagefiles, auf das alle Prozesse zugreifen können. Im Paket layout werden die Darstellungen der einzelnen Activities gespeichert und angepasst, wobei hier auch auf drawable zugegriffen wird, um die unterschiedlichen Elemente und Farben richtig darzustellen.

Das Paket raw beinhaltet die eingelesenen CSV-Dateien mit den Strukturen der Grunddaten, welche verwendet werden sollen.

3.4. Activities der Applikation

Um ein besseres Verständnis der einzelnen Activities und der Funktionsweise zu bekommen, soll das folgende Kapitel die wichtigsten Activities darstellen und die Verwendung erklären.

3.4.1. MainActivity

Die Main Activity besteht aus der Klasse MainActivity, dem XML-File activity_main.xml und dem RecyclerViewAdapterMain. Die Main Activity ist die Hauptplattform der gesamten Applikation und stellt den ersten Berührungspunkt für den User dar. Die folgende Grafik zeigt die Benutzeroberfläche nach dem Start der Applikation.

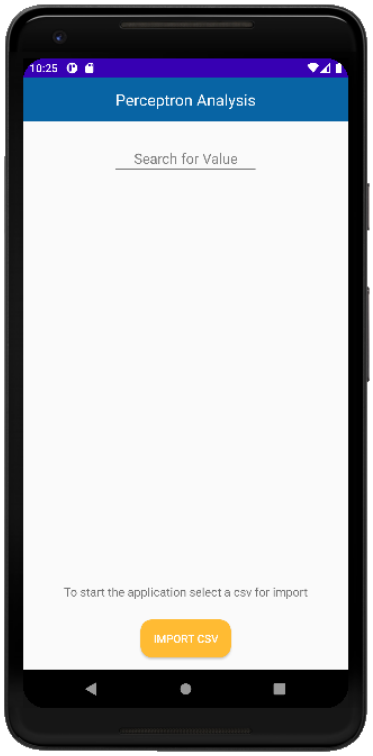


Abbildung 16: MainActivity Startpage

Die Activity zeigt noch nichts da die Daten noch geladen werden müssen, um die Funktionsweise zu starten. Am unteren Rand des Bildschirms wird der User informiert, dass er für eine Verwendung der Applikation zuerst die Daten importieren muss. Hierfür kann der User auf den Button Import CSV drücken und wird weitergeleitet zu einem File-Picker, wo der User eine CSV-Datei auswählen kann.

Wenn der User auf den Button klickt und ein CSV-File auswählt, wird die Activity neu geladen und der User sieht folgende Oberfläche.

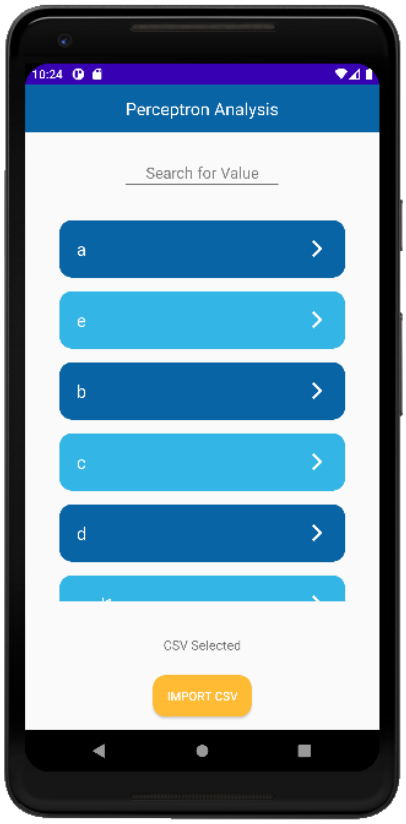


Abbildung 17: MainActivity befüllt

Man sieht nun eine Liste aller eingelesenen Nodes die in dem CSV-File vorhanden sind. Der User kann in dem Textfeld oberhalb der Liste nach einzelnen Einträgen suchen oder die Liste mittels Touchscreen durchscrollen. Zusätzlich dazu ändert sich das Textfeld am unteren Bildschirmrand und gibt den User ein Feedback, dass ein CSV-File selektiert wurde. Der User kann in dieser Phase immer das CSV-File tauschen und ein anderes auswählen, indem er wieder auf den Button klickt und ein anderes File auswählt. Wenn der User auf ein Element in der Liste klickt, wird er weitergeleitet an die NodeActivity.

3.4.2. NodeActivity

Die Node Activity besteht aus der Klasse NodeActivity, dem XML-File activity_node.xml und dem RecyclerViewAdapterNode. Diese Activity beschäftigt sich mit der Darstellung der einzelnen Nodes und den zugehörigen Werten, welche aus dem CSV-File extrahiert wurden. Wenn der User in der MainActivity auf einen

Knotenpunkt klickt, wird er auf die erste Seite der Node Activity weitergeleitet, welche im folgenden Bild zu sehen ist.

Abbildung 18: NodeActivity Startseite

Der User befindet sich nun in der Knotenpunkt Übersicht und bekommt detaillierte Informationen zu dem ausgewählten Knotenpunkt. Es gibt zwei verschiedene Einstellungen, die eine Selektion ermöglichen. In der ersten Ansicht werden immer die höchsten und niedrigsten Werte der importierten Daten angezeigt. In der zweiten Ansicht kann der User alle Werte der Node anzeigen lassen. Diese Unterscheidung ermöglicht es einen schnellen Überblick über eine Node zu bekommen, aber auch auf eine detailliertere Version wechseln zu können.

Der User wird immer informiert welchen Ansichtspunkt er gerade sieht. Dies geschieht, wenn er auf einen anderen Button klickt und somit sich die Farbe des Randes auf schwarz ändert und der User somit direkt sehen kann, welche Auswahl er gerade getroffen hat.

Zusätzlich zu den ersten zwei Sortiermöglichkeiten gibt es drei weitere, die für alle Daten und für die höchsten und niedrigsten gleichermaßen funktioniert. Der User kann

sich durch die Kategorien Weather, Time und Temperature klicken und bekommt dadurch weiter selektierte Daten.

Der Button Weather präsentiert in dem Fall nur Werte, die Informationen zum Wetter beinhalten. Der Button Time stellt nur zeitliche Informationen und Temperature nur Temperaturdaten dar. Durch diese Sortiermöglichkeiten ist es für den User sehr einfach aus jeder Kategorie die höchsten und niedrigsten Werte zu sehen, aber auch wenn notwendig alle Daten zu einer Kategorie einzusehen.

Unter der Liste ist ein weiterer Button mit dem Titel „Icon Description“ dieser Button führt den User zu einer Übersichtsseite, welche die einzelnen Symbole in der Applikation erklären soll.

3.4.3. DescriptionActivity

Die Description Activity besteht aus der Klasse DescriptionActivity, dem XML-File activity_description.xml und dem RecycleViewAdapterElementDescription. Die Activity präsentiert alle Icons, die in der Applikation verwendet werden und soll dem User eine Hilfestellung sein, wenn die Icons nicht selbsterklärend sind. Die folgende Grafik zeigt die Activity.

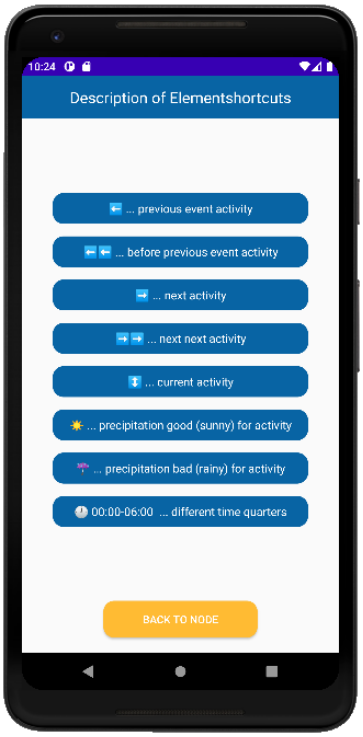


Abbildung 19: IconDescription

Der User sieht eine Liste mit den einzelnen Icons, die verwendet werden und kann nachsehen sollte ein Icon nicht eindeutig sein. Mit dem Button „Back To Node“ kommt der User wieder zurück zur vorherigen Ansicht.

3.5. Import der Ressourcen

Für die Grunddaten des Projektes wurde die bereitgestellte CSV-Datei df_perceptron.csv genommen und das Projekt wurde um diese Struktur herum gebaut. Um diese Daten und die einzelnen Werte für den Leser in Icons zu übersetzen, wurde eine weitere CSV-Datei erstellt, welche es möglich macht, die Abkürzungen der einzelnen Events lesbar zu machen. Diese Datei hat die Bezeichnung mapping.csv und wurde aus dem bereitgestellten Python Projekt extrahiert und die wichtigsten Informationen gebündelt dargestellt. In den folgenden Grafiken sieht man wie die einzelnen Events logisch mit Icons gemappt werden.

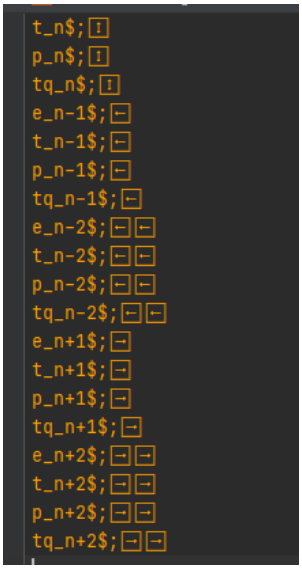


Abbildung 20: CSV Datei Eventinformation

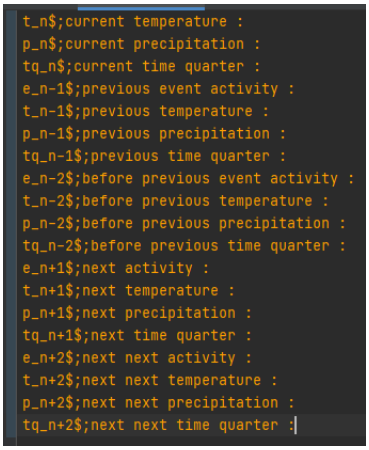


Abbildung 21: CSV Datei Eventinformation detailliert

Die obere Abbildung zeigt die CSV-Datei und die Struktur mittels Icons für die Description der Elemente. Die untere Abbildung ist eine direkte Übersetzung in Sprache. Diese dient zur Veranschaulichung und Erklärung der Icons der oberen Abbildung.

Die CSV-Datei wird automatisch vom System eingelesen und verarbeitet. Diese Struktur ist essenziell für die Logik des Projektes und darf nicht vom User geändert werden.

3.6. Änderungen an der Struktur der Daten

Um Änderungen an der Struktur durchzuführen, muss das Projekt nicht komplett angepasst werden. Sollte das Projekt erweitert werden und es kommen neue Werte hinzu muss in der df_perceptron.csv Datei dies spezifiziert werden. Solange die Werte dem Muster folgen, wie in den Grafiken vom vorherigen Kapitel beschrieben, muss die Applikation nicht erweitert werden. Dies ist ersichtlich aus der folgenden Abbildung.

```
if (valueBefore.contains("t_n")) {
    valueToCheck = changeForTemperature(valueToCheck);
} else if (valueBefore.contains("p_n")) {
    valueToCheck = changeForPrecip(valueToCheck);
} else if (valueBefore.contains("tq_n")) {
    valueToCheck = changeValueForQuarter(valueToCheck);
}
```

Abbildung 22: Funktion ChangePrefix

Dieser Teilausschnitt kommt aus der Funktion convertEventPrefixForNode innerhalb der Klasse Storage. Hier wird der Prefix einer Node konvertiert. Aus der obenstehenden Grafik geht hervor, dass immer der Anfang des Prefix enthalten sein muss. Das heißt konkret, wenn man als Beispiel tq_n+3 hinzufügt, funktioniert die Applikation immer noch. Wird aber ein komplett neues Argument hinzugefügt, muss dies hier geändert werden und eine entsprechende Funktion zur Konvertierung geschrieben werden. Für die bestehenden Kategorien gibt es die Funktionen changeForTemperature, changeForPrecip und changeValueForQuarter. Diese konvertieren den Wert und geben dann die korrekte Zeit, Temperatur oder Precipation zurück.

4. Lessons learned & Probleme bei der Implementierung

Durch das Programmieren und die Bachelorarbeit an sich, traten verschiedene Probleme auf, welche behandelt werden mussten und dadurch wurden neue Erkenntnisse und Wissen gewonnen. Diese werden in diesem Kapitel behandelt und erläutert.

4.1. Leerstring am Anfang eines CSV-Files

Um die Übersetzungen in realer Sprache der einzelnen Events durchzuführen, wurde eine eigene CSV-Datei erstellt und die Daten ausgelesen. Nach dem Auslesen wurde bei der Darstellung der Daten eine Übersetzung durchgeführt. Hierfür gab es eine Funktion translate, die die einzelnen Werte aus der CSV-Datei durchgeht und dann mit dem Wert vergleicht, der übersetzt werden soll und dann die Icons ausgibt. Hier gab es aber einen Bug mit dem ersten Wert aus der CSV-Datei. Nach dem Einlesen des Strings aus der CSV hatte der String immer einen versteckten extra Wert im Speicher. Dieser Wert hatte es unmöglich gemacht, die beiden Werte mit .equals als String zu vergleichen. In der folgenden Grafik sieht man, dass der Count der beiden Werte unterschiedlich ist, das heißt dass die Bit Anzahl unterschiedlich ist, obwohl die Werte identisch sind. Der linke Wert ist der eingelesene Wert. Der Rechte stellt den zu übersetzenden Wert dar. Dieses Problem konnte nur durch einen hardcodierten Vergleich der beiden Werte behoben werden, in diesem Fall t_n, was aber keine ideale Lösung ist.

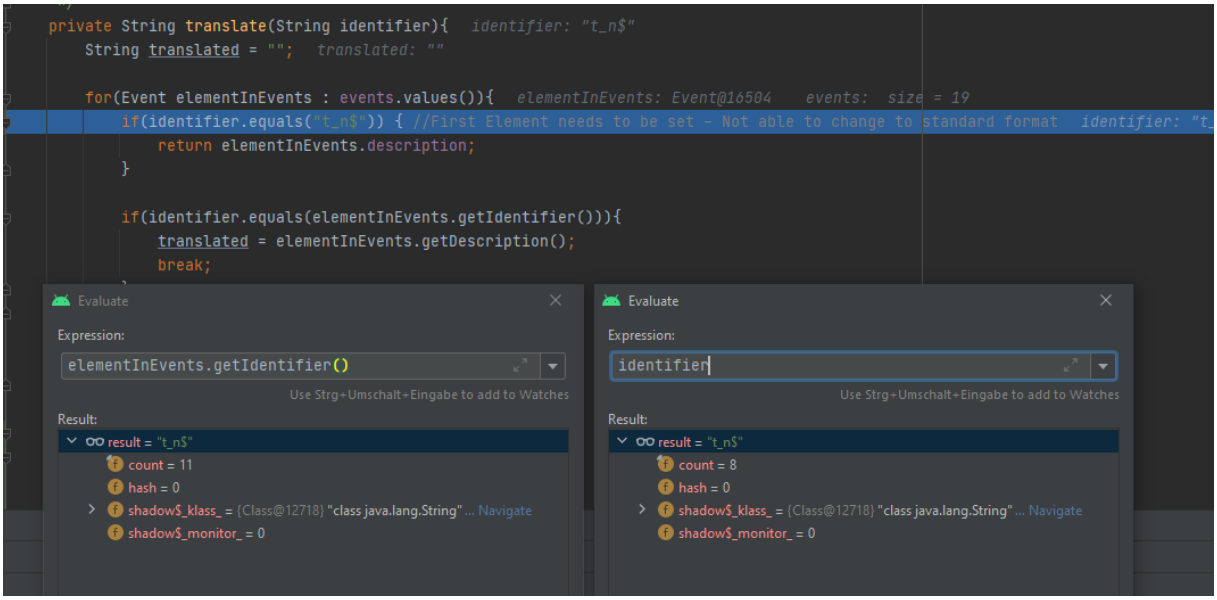


Abbildung 23: Applikation Leerstring Problem

4.2. Variabilität der Grunddaten

Die Grunddaten haben verschiedene Werte, die nicht immer genau der normalen Struktur folgen, für diese Daten wurden Ausnahmen gemacht und eigene Fälle für die Übersetzung kreiert. Für diese Übersetzungen gab es Regelfälle, welche nach den Angaben des Betreuers geregelt wurden und auch fehlerhafte beziehungsweise nicht erwünschte Werte, welche extra behandelt werden mussten. Diese Einschränkungen der Daten führt dazu, dass das Projekt im speziellen das eingelesene CSV-File nicht komplett dynamisch anpassbar sein kann, da die Daten und im speziellen die Event Definition einem gewissen Standard unterliegen und nicht komplett austauschbar gestaltet werden können. Dies wurde in dem Kapitel 3.6 schon besprochen und erklärt.

Hier gab es 3 verschiedene Fälle:

Uhrzeiten abhängig vom Quartal des Tages, Temperaturen des Events, Wetterbedingungen des Events.

4.2.1. Quartalsdaten

Für die Quartalsdaten gibt es 4 definierte Quartale des Tages, die auftreten können. Hierzu mussten die Werte übersetzt werden und für den User lesbar werden. Es gab aber einen Standardfall für alle Werte, die nicht dem Entsprechen. Diese Aufgabe hat die Funktion changeValueForQuarter durchgeführt. Hier wurden die Standarddaten durch die Uhrzeiten ersetzt und die Werte, welche nicht auf dem definierten Spektrum liegen wurden durch ? ersetzt. Der in der Grafik verwendete StringBuilder erstellt ein Icon für die Uhrzeit und stellt dieses Icon dann dar.

```
private String changeValueForQuarter(String value) {
    switch (value) {
        case "q1":
            return new StringBuilder().appendCodePoint(0x1F55B).toString() + "00:00-06:00";
        case "q2":
            return new StringBuilder().appendCodePoint(0x1F55B).toString() + "06:00-12:00";
        case "q3":
            return new StringBuilder().appendCodePoint(0x1F55B).toString() + "12:00-18:00";
        case "q4":
            return new StringBuilder().appendCodePoint(0x1F55B).toString() + "18:00-24:00";
        default:
            return new StringBuilder().appendCodePoint(0x1F55B).toString() + " ? ";
    }
}
```

Abbildung 24: Applikation Quartalsdaten

4.2.2. Temperaturdaten

Bei den Temperaturdaten gibt es auch wie bei den Quartalsdaten einen Standardfall, der eintritt, wenn eine Temperatur im Format 00.0-00.0 ist. Die Nullen sind im Standardfall die richtigen Temperaturen der Daten. Hier gibt es aber auch mehrere extra Fälle, die durch das Trainieren des Perceptron auftreten. Ein Beispiel wäre der Wert -91.0. Dieser Wert ist keine Temperatur, sondern tritt nur auf, wenn das Perceptron trainiert wird und sollte durch ein ? ersetzt werden. Hierzu wurde sich eine leichte Methode überlegt, dass die Länge des Strings immer mindestens 6 sein muss um die Struktur 00.0-00.0 zu gewährleisten. In der Grafik ist der Teil \uD83C\uDF21 enthalten. Dieser Teil ist der String Code für das Temperatur Zeichen in Android.

```
private String changeForTemperature(String value) {
    return value.length() <= 6 ?
            "\uD83C\uDF21 ?"
            :
            value + "° celcius";
}
```

Abbildung 25: Applikation Temperaturdaten

4.2.3. Wetterbedingungen

Auch bei den Wetterdaten gibt es einen Standardfall. Der Wert 0.0 beschreibt das Wetter als ohne Niederschlag und der Wert 1.0 beschreibt das Wetter mit Niederschlag. Hier kommt es aber auch zu anderen Werten, die nicht in dieses Spektrum passen. Genau wie bei den Temperaturen ist dies für das Trainieren des Perceptron notwendig. Hier wurde auch eine extra Funktion gebaut, die es möglich macht, diese Standardfälle zu filtern und Werte, die für das Training des Perceptrons verwendet werden gegen ? zu tauschen, da diese nicht relevant für den User sind.

```
private String changeForPrecip(String value) {
    switch (value) {
        case "0.0":
            return "☀";
        case "1.0":
            return "☔";
        default:
            return "?";
    }
}
```

Abbildung 26: Applikation Wetterdaten

4.3. Perceptron Darstellung

Die Darstellung des Perceptrons als dynamische Grafik in der Android Applikation war vom Betreuer als sekundäres Ziel gewünscht, konnte aber nicht erfüllt werden. Es wurden etliche Recherchen durchgeführt, um ein Framework zu finden, welches es möglich macht das trainierte Perceptron darzustellen. Es gibt zwar für Java verschiedene Frameworks, die es möglich machen Perceptron in Graphen darzustellen, aber eine wirkliche Darstellung mit Punkten und visuellen Pfeilen gibt es nicht auf dem aktuellen Markt. Diese Graphen sind klassische Linien oder Balken Graphen, die den Zweck nicht erfüllen würden. Um eine andere Methode auszuprobieren, wurde versucht mittels Canvas die Punkte darzustellen und die notwendigen Pfeile der Verbindungen der einzelnen Knoten zu zeichnen. Diese Grafiken waren aber für den kleinen Bildschirm des Telefons nicht geeignet und wurden aus diesen Gründen weggelassen. Zusätzlich zu der Lesbarkeit wurde auch die Komplexität mit mehreren Knoten viel zu unübersichtlich und nicht anwendbar.

5. Resümee

Die Bachelorarbeit war eine gute Gelegenheit, neue Informationen und Arbeitsweisen im Bereich von Android zu entdecken. Die größten Herausforderungen waren die anfänglichen Nachforschungen für die Struktur und Funktionsweisen des Perceptron und der Analyse der Grunddaten. Zusätzlich dazu gab es mehrere Probleme bei der Modularität des Projektes und der Applikation, da die Daten sehr stark definiert sind und nicht viel Änderungen zulassen. Das Projekt war bis auf die visuelle Darstellung des Perceptron ein Erfolg und hat eine neue Begeisterung für Android und Appentwicklung im generellen geweckt.

Literaturverzeichnis

Aalst, W. van der. (2014). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Berlin Heidelberg.

Aalst, W. M. P. van der. (2016). *Process Mining: Data Science in Action*. Springer.

Murphy, K. P. (2012). *Machine Learning—A Probabilistic Perspective*. 26.

Nitta, T. (2003). Solving the XOR problem and the detection of symmetry using a single complex-valued neuron. *Neural Networks*, *16*(8), 1101–1105. https://doi.org/10.1016/S0893-6080(03)00168-0

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Sabry, F. (2023). *Perceptrons: Fundamentals and Applications for The Neural Building Block*. One Billion Knowledgeable.

Trask, A. W. (2019). *Neuronale Netze und Deep Learning kapieren: Der einfache Praxiseinstieg mit Beispielen in Python*. MITP-Verlags GmbH & Co. KG.